Ensembles of Models for Automated Diagnosis of System Performance Problems

Steve Zhang[2], Ira Cohen[1] , Moises Goldszmidt[1], Julie Symons[1], Armando Fox[2]

[1] {ira.cohen, moises.goldszmidt, julie.symons}@hp.com, Hewlett Packard Research Labs

[2] {steveyz, fox}@cs.stanford.edu, Stanford University

Abstract

Violations of service level objectives (SLO) in Internet services are urgent conditions requiring immediate attention. Previously we showed [1] that Tree-Augmented Bayesian Networks or TAN models are effective at identifying which low-level system properties were correlated to high-level SLO violations (the *metric attribution* problem) under stable workloads. In this paper we extend our approach to adapt to changing workloads and external disturbances by maintaining an ensemble of probabilistic models, adding new models when existing ones do not accurately capture current system behavior. Using realistic workloads on an implemented prototype system, we show that the ensemble of TAN models captures the performance behavior of the system accurately under changing workloads and conditions. We fuse diagnoses from the ensemble of models to identify likely causes of the performance problem, with results comparable to those produced by an oracle that continuously changes the model based on advance knowledge of the workload. The cost of inducing new models and managing the ensembles is negligible, making our approach both immediately practical and theoretically appealing.

Keywords: Automated diagnosis, self-healing and self-monitoring systems, statistical induction and Bayesian Model Management.

1 Introduction

A key concern in contemporary highly-available Internet systems is to meet specified service-level objectives (SLO's), which are usually expressed in terms of high-level behaviors such as average response time or request throughput. SLO's are important for at least two reasons. First, many services are contractually bound to meet SLO's, and failure to do so may have serious financial and customer-backlash repercussions. Second, although most SLO's are expressed as measures of performance, there is a fundamental connection between SLO monitoring and availability, both because availability problems often manifest early as performance problems and because understanding how

different parts of the system affect availability is a similar problem to understanding how different parts of the system affect high-level performance metrics.

However, these systems are complex. A three-tier Internet service includes a Web server, an application server and possibly a database or other persistent store; large services replicate this arrangement many-fold, but the complexity of even a non-replicated service encompasses various subsystems in each tier. An enormous number of factors, including performance of individual servers or processes, variation in resources used by different types of user requests, and temporary queueing delays in I/O and storage, may all affect a high-level metric such as response time. Without an understanding of which factors are actually responsible for SLO violation under a given set of circumstances, repair would have to proceed blindly, e.g. adding more disk without knowing whether disk access is really the SLO bottleneck.

We refer to this as *metric attribution*: under a given set of circumstances, which low-level metrics are most correlated with a particular SLO violation? Note that this is subtly different from root-cause diagnosis, which usually requires domain-specific knowledge. Metric attribution gives us the subset of metrics that are most highly correlated with a violation, and in practice this often does point to the root cause, but metric attribution does not require domain-specific knowledge as diagnosis would.

In [1] we explored a pattern recognition approach that builds statistical models for metric attribution, allowing an operator to focus his attention on a small number of possible repairs. Using complex workloads that stress (but do not saturate) system performance, we demonstrated that while no single low-level sensor is a good predictor of SLO violation, a tree-augmented naive Bayesian network (TAN) model [3] captures 90–95% of the patterns of SLO behavior by combining a small number of low-level sensors, typically 3–8. The experiments also hinted at the need for adaptation, since under very different conditions, different metrics and different thresholds on the metrics are selected as significant by these models. This is unsurprising since metric attribution may vary considerably depending on changes in the workload due to surges or rebalancing (due to transient node failure), changes in the infrastructure (including software and hardware revisions and upgrades), faults in the hardware, and bugs in the software, among other factors. In this paper we present methods to enable adaptation to these changes by maintaining an ensemble of models. As the system is running we periodically induce new models and enhance the ensemble only if the new models capture a behavior that no existing model captures. The problem of diagnosis under these conditions is thereby reduced to the problem of managing this ensemble: for the best diagnosis of a particular SLO problem, which model(s) in the ensemble are relevant, and how can we combine information from those models regarding the system component(s) implicated in the SLO violation?

We test our methods using a 3-tier web service, and the workloads in [1] augmented with other workloads some of which simulate performance problems by having an external application load the CPU of different components in

the infrastructure. The experiments demonstrate that:

- We can successfully manage an ensemble of TAN models to provide adaptive diagnosis of SLO violations under changing workload conditions.
- Using a subset of models in the ensemble yields more accurate diagnosis than using a single monolithic TAN model trained on all existing data. That is, the inherent multi-modality characterizing the relationship between high-level performance and low-level sensors is captured more accurately by an ensemble of models each trained on one "mode" than by a single model trying to capture all the modes.
- Our techniques for managing the ensemble and selecting the right subset for diagnosis are inexpensive and efficient enough to be used in real time.

The rest of this paper is organized as follows. Section 2 reviews our previous results and describes our metrics of model quality and how they are computed. Section 3 describes how we extend that work to manage an ensemble of models; it describes how new models are induced, how the analyzer decides whether to add them to the ensemble, and how models are selected for problem diagnosis and metric attribution. In sections 4 and 5 we describe our experimental methodology and results. Section 6 places our contribution in the context of related work, and section 7 concludes.

2 Background and Review of Previous Results

Consider a stimulus-driven request/reply Internet service whose low-level behaviors (CPU load, memory usage, etc.) we can measure at frequent intervals, say 15 to 30 seconds. For each successfully completed request, we can determine whether the system is in compliance or non-compliance with a specified service-level objective (SLO), for example, whether the total service time did or did not fall below a specified threshold.

In [1] we cast the problem of inducing a metric-attribution model as a pattern classification problem in supervised learning. Let $S_t \in \{s^+, s^-\}$ denote whether the system is in compliance (s^+) or noncompliance (s^-) with the SLO at time t. Let $\vec{M}_t = [m_1, ..., m_n]$ denote the values of n collected metrics at time t (we will omit the subindex t when the context is clear). The pattern classification problem consists of inducing or learning a classifier function $\mathcal{F} : \vec{M}_t \rightarrow \{s^+, s^-\}$. In order to determine a figure of merit for the function $\mathcal{F}$ it is customary to estimate the probability that when $\mathcal{F}$ applied to some $\vec{M}_t$ it yields the correct value of S (s^+ or s^-). In our domain of application where it is expected that one class s^+ statistically dominates the other, it is common practice to use *balanced accuracy*

(BA), which averages the probability of correctly identifying compliance with the probability of detecting a violation.

$$BA = (0.5) \times [P(s^- = \mathcal{F}(\vec{M})|s^-) + P(s^+ = \mathcal{F}(\vec{M})|s^+)] \tag{1}$$

In order to achieve the maximal BA of 1.0, $\mathcal{F}$ must perfectly classify both SLO violation and SLO compliance, whereas a trivial classifier that always predicts the majority class has the minimum BA of 0.5.

To implement the function $\mathcal{F}$, we first use a probabilistic model that represents the joint distribution $P(S,\vec{M})$ – the distribution of probabilities for the system state and the observed values of the metrics. From the joint distribution we compute the conditional distribution $P(S|\vec{M})$ and the classifier uses this distribution to evaluate whether $P(s^-|\vec{M}) > P(s^+|\vec{M})$. Using the joint distribution enables us to reverse $\mathcal{F}$, so that the influence of each metric on the violation of an SLO can be quantified with a sound probabilistic semantics.

In our representation of the joint distribution, we arrive at the following functional form for the classifier as a sum of terms, each involving the probability that the value of some metric m_i occurs in each state given the value of the metrics m_{p_i} on which m_i depends (in the probabilistic model):

$$\sum_{i=1}^{n} \log[\frac{P(m_i|m_{p_i},s^-)}{P(m_i|m_{p_i},s^+)}] + \log\frac{P(s^-)}{P(s^+)} > 0 \tag{2}$$

From Eq. 2 we see that a metric, *i*, is implicated with an SLO violation if $log[\frac{P(m_i|m_{p_i},s^-)}{P(m_i|m_{p_i},s^+)}]$, also known as the loglikelihood difference (for metric *i*), is greater than 0. Analyzing which metrics are implicated with an SLO violation catalogs each type of SLO violation according to the metrics and values that had a positive loglikelihood difference. Furthermore, the *strength* of each metric's influence on the classifier's choice is given from the actual value of the loglikelihood difference.

We rely on Bayesian networks to represent the probability distribution $P(S,\vec{M})$. Bayesian networks are computationally efficient representational data structures for probability distributions [2]. The use of Bayesian networks as classifiers has been studied extensively [3]. Furthermore, in [1] as well as in this paper we use the a class of Bayesian networks know as Tree Augmented Naive Bayes (TAN) models. The benefits of TAN and its performance in pattern classification is studied and reported in [3]. The key results from our work [1] on using TAN classifiers to model the performance of a 3-tier web service include:

- Combination of metrics are significantly more predictive of SLO violations than individual metrics. Moreover, different combinations of metrics and thresholds are selected under different conditions. This implies that most problems are too complex for simple "rules of thumb" (e.g., only monitor CPU utilization).

- Small numbers of metrics (typically 3-8) are sufficient to predict SLO violations accurately. In most cases the

selected metrics yield insight into the cause of the problem and its location within the system.

- Although multiple metrics are involved, the relationships among these metrics are relatively simple. Thus, TAN models are highly accurate: in typical cases, the models yield a BA of 90% - 95%.

Based on these findings, and the fact that TAN models are extremely efficient to induce, represent and evaluate, we hypothesized that the approach could be extended to enable a strategy of adaptation based on periodically inducing new models and managing an ensemble of models. This leads to the following questions, which are addressed and validated in the next sections:

1. Can we achieve better accuracy by managing multiple models as opposed to refining a single model? (Section 3.1)

2. If so, how do we induce new models and how do we decide whether a new model is needed to capture a behavior that is not being captured by any existing model? (Section 3.2)

3. When different model(s) are correct in their classification of the SLO state, but differ in metric attribution, which model should we "believe"? (Section 3.3)

3 Managing an Ensemble of Models

We explore the adaptation issue by extending our previous work to manage an *ensemble* of classifiers. We sketch our approach as follows. We treat the workload as a sequence of time periods. During each period, we score the BA of existing models in the ensemble against the workload of that period, and compare this to the BA of a new model induced during that period. Based on this comparison, the new model is either added to the ensemble or discarded.

Whenever the system is violating its SLO (in our case end-to-end request service time), we use the Brier score [4, 5] to select the most relevant model from the ensemble for metric attribution. The Brier score is a metric related to the mean-squared-error often used in statistical fitting as a measure of model goodness. The analyzer then applies Eq. 2 to this model to extract the list of low-level metrics most likely to be implicated with the violation. This information can then be used by an administrator or automated repair agent.

In the rest of this section we give the rationale for this approach and describe the specifics of how new models are induced and how the model ensemble is managed.

3.1 Multiple Models vs. Single Model

The observation that changes in workload and other conditions leads to changes in metric-attribution relationships has been observed in real Internet services [6] and in the experiments in [1]. In the context of probabilistic models in general, and TAN models in particular, there are various ways to handle such changes.

The first is to learn a single monolithic TAN model that attempts to capture all variations in workload and other conditions. While we bring a more detailed analysis of this approach in Section 5, the following experiment demonstrates its shortcomings. We use data collected on a system with two different workloads (described in Sections 4.2.1 and 4.2.2), one that mimics increasing but "well-behaved" activity (e.g. when a site transitions from a low-traffic to a high-traffic service period) and the second mimics unexpected surges in activity (e.g. a breaking news story). A single classifier trained on both workloads had a BA of 72.4% (22.6% false alarm rate, 67.4% detection). But by using two models, one trained on each workload, we achieve a BA of 88.4% (false alarm rate 13%, detection rate 90%). Note that the improved BA results from improvements in *both* detection rate and false alarm rate.

A second approach, trying to circumvent the shortcomings of the single monolithic model, is a single model that adapts with the changing conditions and workloads. Adapting the TAN model in such a way is fairly simple: all that is needed is the update of means and covariance matrices (as we are using Gaussians to represent the interaction between the metrics) which can be performed incrementally (see for example [7]), and every so often we simply perform a metric search and run the algorithms require to induce a TAN model [3]. There are many issues to be considered with respect to this strategy regarding the aging of the statistics which in turn will affect the speed by which the TAN model adapts to the changes in the data. The biggest shortcoming of such an approach is in the fact that the single model does not have "long term" memory, that is, the model can only capture short term system behavior, forcing it to relearn conditions that might have occurred already in the past.

In the third approach, which is the one taken in this paper, adaptation is addressed by using a scheduled sequence of model inductions, keeping an ensemble of models instead of a single one. The models in the ensemble serve as a memory of the system in case problems reappear. In essence, each model in the ensemble is a summary of the data used in building it. We can afford such an approach since the cost, in terms of computation and memory, of learning and managing the ensemble of models is relatively negligible: SLO measures are typically computed at one to five minutes in commercial products (e.g., HP's OpenView Transaction Analyzer), while learning a model in our current implementation takes about 9-10 seconds, evaluating a model in the ensemble takes about 1 msec and storing a model involves keeping at most 20 floating point numbers[1].

An ensemble of models raises two main issues: First, what is the appropriate number of samples (or window

[1]These performance figures were on a Intel Pentium 4, 2.0GHz laptop, initial benchmarks of the model learning and testing with a new Java implementation indicate an order of magnitude improved performance

size) for inducing a new model and, second, once we have a violation how do we fuse the information provided by the various models. These are the questions we address in the next two sections.

3.2 Inducing New Models and Updating the Model Ensemble

We use a standard TAN induction procedure [3], augmented with a process called *feature selection*, for selecting the subset of metrics that are most relevant to modeling the patterns and relations in the data. The feature selection process provides two advantages. First, it discards metrics that appear to have little impact on the SLO, allowing the human operator to focus on a small set of candidates for diagnosis. Second, it reduces the number of data samples needed to induce robust models. This is known as the dimensionality problem in pattern recognition and statistical induction: the number of data samples needed to induce *good* models increases exponentially with the dimension of the problem, which in our case is the number of metrics in the model (which influences the number of parameters). The number of data samples needed to induce a viable model in turn influences the adaptation process. The feature selection problem is essentially a combinatorial optimization problem which is usually solved using heuristic search; in this paper we use a beam search algorithm, which provides some robustness to local minima [8].

Models are induced periodically over a dataset D composed as in Section 2 of a vector of n metrics at time t, $\vec{M}_t = [m_1, ..., m_n]$ and the corresponding labels over the SLO state. Once the model is induced, we estimate its balanced accuracy (Eq. 1) on the data set D using tenfold crossvalidation [9]. We also compute a confidence interval around the new BA score. If the new model's BA is statistically significantly better than that of all models in the existing ensemble, the new model is added to the ensemble; otherwise it is discarded.

There are two inputs to the ensemble-management algorithm: the number of samples in the window, and the minimum number of samples per class. Choosing too many samples may increase the number of patterns that a single model tries to capture (approaching the single-model behavior described previously) and result in a TAN model with more parameters. Too few may result in a non-robust model and overfitting of the data. Lacking closed-form formulas that can answer these questions for models such as TAN and for potentially complex workloads such as those we emulate, we follow the typical practice in machine learning of using an empirical approach. We use *learning surfaces* to characterize minimal data requirements for the models in terms of number of data samples required and proportions of SLO violation versus compliance as described in Section 5.3. This provides an approximate estimate of the basic data requirements. We then validate those requirements in the experiments we perform.

Algorithm 1 describes in detail the algorithm for managing the ensemble of models.

Algorithm 1 Learning and Managing an Ensemble of Models

```
Input: TrainingWindowSize and Minimum Number of Samples Per Class
Initialize Ensemble to {ϕ} and TrainingWindow to {ϕ}
for every new sample do
  add sample to TrainingWindow
  if TrainingWindow has Minimum Number of Samples Per Class then
    Train new Model M on TrainingWindow (sec. 3.2)
    Compute accuracy of M using crossvalidation.
    For every model in Ensemble compute accuracy on TrainingWindow.
    if accuracy of M is significantly higher than the accuracy of all models in the Ensemble then
      add new model to Ensemble.
    end if
  end if
  Compute Brier score (Eq. 3) over TrainingWindow for all models in Ensemble (sec. 3.3)
  if system state for new sample is SLO violation then
    Perform metric attribution using Eq. 2 over metrics of model with lowest Brier score (Winner Takes All).
  end if
end for
```

3.3 Inference: Metric Attribution and Ensemble Accuracy

When using a single model, accuracy is computed by counting the number of cases in which the model correctly predicts the known SLO state and metric attribution is performed by applying Eq. 2: every metric m_i for which the log-likelihood ratio difference is positive is a metric whose value is consistent with the state of SLO violation. To compute accuracy and metric attribution using an ensemble we follow a *winner take all* strategy: we select the "best" model from the ensemble, and then proceed as in the single-model case.

Although the machine learning literature describes methods for weighted combination of models in an ensemble to get a single prediction [10, 11, 12], our problem is complicated and differs from most cases in the literature in that the workloads and system behavior are not stationary. As we have observed, this leads to different causes of performance problems at different times even though the high level observable manifestation is the same (violation of the SLO state). We find that given a suitable window size (number of samples) we are able to obtain fairly accurate models to characterize windows of nonstationary workload; hence we expect that the simpler winner-take-all approach should provide good overall accuracy.

For each model in the ensemble we compute its Brier score [4] over a short window of past data, $D = \{d_{t-w}, ..., d_{t-1}\}$, where t is the present sample, making sure D includes samples of both SLO compliance and non-compliance. The Brier score is the mean squared error between a model's probability of the SLO state given the current metrics and the actual value of the SLO state, i.e., for every model in the ensemble, Mo_j:

$$BS_{Mo_j}(D) = \sum_{k=t-1}^{t-w} [P(s^- | \vec{M} = \vec{m}_k; Mo_j) - I(s_k = s^-)]^2, \tag{3}$$

where $P(s^- | \vec{M} = \vec{m}_i; Mo_j)$ is the probability of the SLO state being in violation of model j given the vector of metric

measurements and $I(s_k = s^-)$ is an indicator function, equal to 1 if the SLO state is s^- and 0 otherwise, at time k. For a model to receive a good Brier score (best score being 0), the model must be both correct and confident (in terms of the probability assessment of the SLO state) in its predictions.

An alternative, often used in the literature, to the Brier score, is the accuracy of the models in the ensemble. However, we observed in our experiments that the use of accuracy as the weight consistently produced worse results compared to the Brier score. Intuitively, explaining this observation is the differentiability provided by the Brier score over the accuracy measure, which is important in our case since we use small values of the window size w to compute the weight of each model.

Note that we are essentially using a set of models, to capture the drift in the relationship between the low level metrics and performance as defined by the SLO state. The Brier score is used as a proxy for modeling the change in the probability distribution governing these relationship by selecting the model with the minimal mean squared error (in the neighborhood) of an SLO violation.

4 Experimental Methodology

We validate our methods on a three-tier system running an Internet web service. We excite the system using complex workloads, inducing performance problems that manifest as high response time, thus violating the system's SLO. We empirically designed our workloads to stress, but not overload, our system during abnormal periods; once we determined the point at which our workload saturated the system (regardless of the underlying cause)[2], we then aimed for steady-state workload at 50–60% of this threshold, as is typical for production systems based on J2EE [13, 14].

For analysis, we collect both low level system metrics from each tier (e.g., CPU, memory, disk and network utilizations) and application level metrics (e.g., response time and request throughput). The following subsections describe the testbed and workloads we use to excite the system.

4.1 Testbed

Figure 1 depicts our experimental three-tier system, the most common configuration for medium and large Internet services. The first tier is the Apache 2.0.48 Web server, the second tier is the application itself, and the third tier is an Oracle 9iR2 database. The application tier runs PetStore, an e-commerce application freely available from The Middleware Company (we used Petstore version 2.0). This application is provided as an example of how to build applications for the Java 2 Enterprise Edition (J2EE) application environment. We tuned the deployment descriptors, config.xml, and startWebLogic.cmd in order to scale to the transaction volumes reported in the results.

[2]By saturation we mean the point from which response time grows unbounded with a constant workload.

J2EE applications must run on middleware called an application server; we use WebLogic 7.0 SP4 from BEA Systems, one of the more popular commercial J2EE servers, with Java 2 SDK 1.3.1 from Sun.

Each tier (Web, J2EE, database) runs on a separate HP NetServer LPr server (500 MHz Pentium II, 512 MB RAM, 9 GB disk, 100 Mbps network cards, Windows 2000 Server SP4) connected by a switched 100 Mbps full-duplex network. Each machine is instrumented with HP OpenView Operations Agent 7.2 to collect system-level metrics at 15-second intervals, which we aggregate to one minute intervals, each having 124 metrics. A load generator called *httperf* offers load to the service over a sequence of execution intervals. An SLO indicator processes the Apache web server logs to determine SLO compliance over each interval, based on the average server response time for requests in the interval.

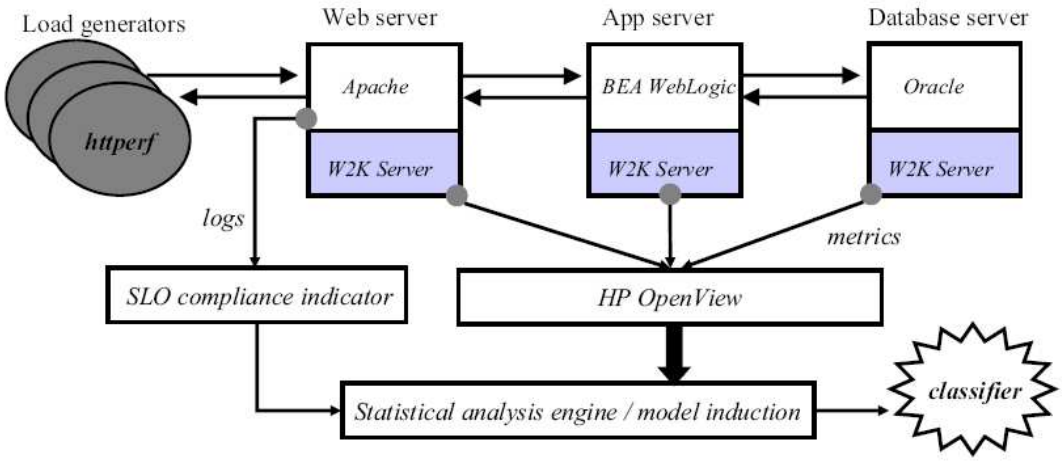


Figure 1: Our experimental system, featuring a commonly used three-tier internet service.

4.2 Workloads

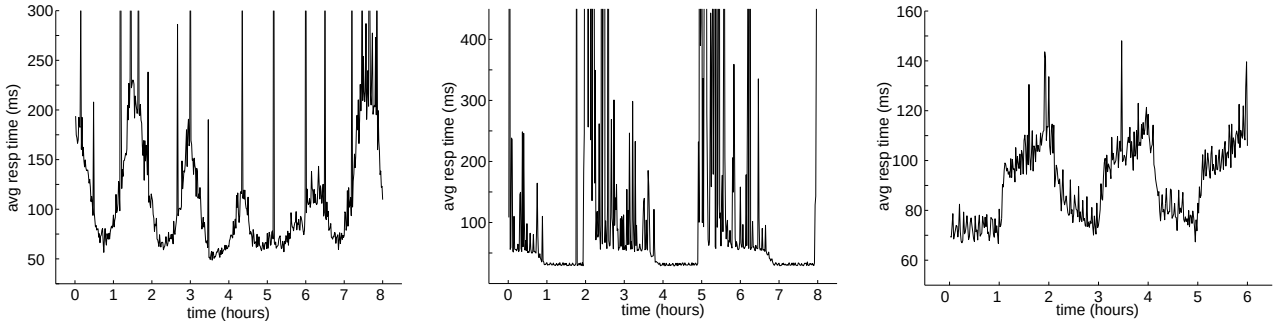
All of our workloads were created to exercise the three-tiered webservice in different ways. We rejected the use of standard synthetic benchmarks such as TPC-W that ramp up load to a stable plateau in order to determine peak throughput subject to a constraint on mean response time. Such workloads are not sufficiently rich to mimic the wide range of system conditions that might occur in practice.

We designed the workloads to exercise our algorithms by providing it with a wide range of $(\vec{M},\vec{P})$ pairs, where $\vec{M}$ represents a sample of values for the system metrics and $\vec{P}$ represents a vector of application-level performance measurements (e.g., response time and throughput). Of course, we cannot directly control either $\vec{M}$ or $\vec{P}$; we control only the workload submitted to the system under test. We wrote simple scripts to generate session files for the httperf

workload generator, which allows us to vary the client think time and the arrival rate of new client sessions.

4.2.1 RAMP: Increasing Concurrency

For this experiment we gradually increase the number of concurrent client sessions. We add an emulated client every 20 minutes up to a limit of 100 total sessions. Individual client request streams are constructed so that the aggregate request stream resembles a sinusoid overlaid upon a ramp. The average response time of the web service in respond to this workload is depicted in Figure 2. Each client session follows a simple pattern: go to main page, sign in, browse products, add some products to shopping cart, check out, repeat. P_b is the probability that an item is added to the shopping cart given that it has just been browsed; P_c is the probability of proceeding to the checkout given that an item has just been added to the cart. These probabilities vary sinusoidally between 0.42 and 0.7 with periods of 67 and 73 minutes, respectively. This experiment, as well as the that described in the next section, was used to validate our earlier work. We felt both would be useful for demonstrating some key properties of our current work as well.



(a) (b) (c)

Figure 2: Relevant sequences of average web server response time over 1-minute windows when the test system was subjected to the (a) RAMP workload (b) BURST workload (c) BUYSPREE workload

4.2.2 BURST: Background + Step Function

This run has two workload components. The first *httperf* creates a steady background traffic of 1000 requests per minute generated by 20 clients. The second is an on/off workload consisting of hour-long bursts with one hour between bursts. Successive bursts involve 5, 10, 15, etc. client sessions, each generating 50 requests per minute. The intent of this workload is to mimic sudden, sustained bursts of increasingly intense workload against a backdrop of moderate activity. Each step in the workload produces a different plateau of workload level, as well as transients during the beginning and end of each step as the system adapts to the change.

Workload	Normal Condition	Abnormal Condition
BUYSPREE	$P_b = 0.7; P_c = 0.7; X_{db} = 0; X_{app} = 0$	$P_b = 1.0; P_c = 1.0; X_{db} = 0; X_{app} = 0$
DBCPU	$P_b = 0.7; P_c = 0.7; X_{db} = 0; X_{app} = 0$	$P_b = 0.7; P_c = 0.7; X_{db} = 0.3; X_{app} = 0$
APPCPU	$P_b = 0.7; P_c = 0.7; X_{db} = 0; X_{app} = 0$	$P_b = 0.7; P_c = 0.7; X_{db} = 0; X_{app} = 0.3$

Table 1: Summary of new workloads. P_b is the probability an item is added to cart if it is browsed. P_c is the probability of proceeding to checkout given an item is added to cart. X_{db} is the extra CPU load placed on the database server. X_{cpu} is the extra CPU load placed externally on the application server.

4.2.3 BUYSPREE: Increasing Load Without Increasing Concurrency

In this experiment, a new client session is added every 2 minutes. However, unlike in RAMP, each sessions only lasts for slightly more than 50 minutes. This means that after a short initial ramp up period, there will always be approximately 25 concurrent sessions. Also, after the first two hours, this experiment alternates between two periods of operations which we will call *normal* and *abnormal.* Each period lasts a single hour. The parameters $P_b$and $P_c$are set to 0.7 with no variance during the normal periods and are set to 1.0 with no variance during the abnormal periods. The normal-abnormal cycle is repeated 3 times. The resulting average user response time is shown in Figure 2.

4.2.4 DBCPU & APPCPU: External Contention for Server CPU

These two experiments are identical to BUYSPREE except for one important change. During the abnormal periods, P_b and P_c remain at 0.7, but an external program is executed on either the database server (DBCPU) or the application server (APPCPU) that takes away 30% of that server's CPU cycles for the duration of the period. This external program can be compared to a virus scanning process that wakes up every once in a while, except that it uses very little memory and no network resources. The normal-abnormal cycle is repeated 4 times for this experiment and resulting user response times are similar to that in BUYSPREE. The parameters of these workloads are summarized in Table 1.

5 Experimental Results

We report results of our methods using the data collected from all the workloads described above. We compare these results to: (a) a single model trained using all the experimental data, (b) a method that has the information on when exactly there were workloads changes among the five workloads described above, allowing to induce specific TAN models for each of the five workloads and also invoke the appropriate model during testing. This last method is clearly unrealistic, as such information is not available in online settings, but it is useful to compare our method to such an optimistic method.

We learn the ensemble of models as described in Algorithm 1 using a subset of the available data. The subset

of data are the first 4-6 hours of each workload. We keep the last portion of each workload as a test (validation) set. Overall, the training set is 28 hours long (1680 samples) and the test set is 10 hours long (600 samples). The training set starts with alternating two hours sequences of DBCPU, APPCPU, and BUYSPREE, with DBCPU and APPCPU represented three times and BUYSPREE twice. This is followed by six hour sequences of data from RAMP and BURST. The result is a training set which has five different workload conditions some of which are repeated several times. Testing on the test set with the ensemble amounts to the same steps as in Algorithm 1, except for the fact that the ensemble is not initially empty (but has all the models trained on the training data) and no new models are added at any point. The accuracies and metric attribution provided by this testing procedure show how generalizable the models are on unseen data with similar types of workloads. The results of these tests are described below.

5.1 Accuracy

Table 2 shows the balanced accuracy (BA), false alarm and detection rates, measured on the validation data, of our approach compared to the single model and the set of models trained on each of the five workloads. We present results for the ensemble of models with three different training window size (80, 120 and 240 samples). We see that for all cases: (a) the ensemble's performance with any window size is significantly better than the single-model, (b) the ensemble's performance is robust to wide variations of the training window size, and (c) the ensemble's performance is slightly better, mainly in terms of detection rates, compared to the set of five individual models trained on each of the workload conditions we induced.

The third observation is at first glance puzzling, as the set of models trained specifically for each workload should be the best performers. However, some of the workloads on the system were quite complex, e.g., BURST has a ramp up of number of concurrent sessions over time and other varied conditions. This complexity is further characterized in Section 5.3, where we see that it takes many more samples to capture patterns of the BURST workload, and with lower accuracy compared to the other workloads. The ensemble of models, which is allowed to learn multiple models for each workload, is better able to characterize this complexity than the single model trained with the entire dataset.

The table also shows the total number of metrics included in the models and the average number of metrics attributable to specific instances of SLO violations (as discussed in section 3.3). The ensemble chooses many more metrics overall because models are trained independently of each other, which suggests that there is some redundancy among the different models in the ensemble. However, only a handful of metrics are attributed to each instance of an SLO violation, therefore attribution is not affected by the redundancy.

Most of the single model's false alarms occur for the BUYSPREE workload. By observing the metric attribution

	# metrics chosen	avg # attr metrics	BA	FA	Det
Ensemble W80	64	2.3	95.67 ±0.81	4.19± 1.12	95.53± 1.21
Ensemble W120	52	2.5	95.12 ±0.86	4.84± 1.2	95.19± 1.23
Ensemble W240	33	3.7	94.68 ±0.90	5.48± 1.27	94.85± 1.27
Workload specific models	9	2	93.62±0.97	4.51± 1.16	91.75±1.58
Single model	4	2	86.10± 1.38	21.61 ± 2.31	93.81± 1.38

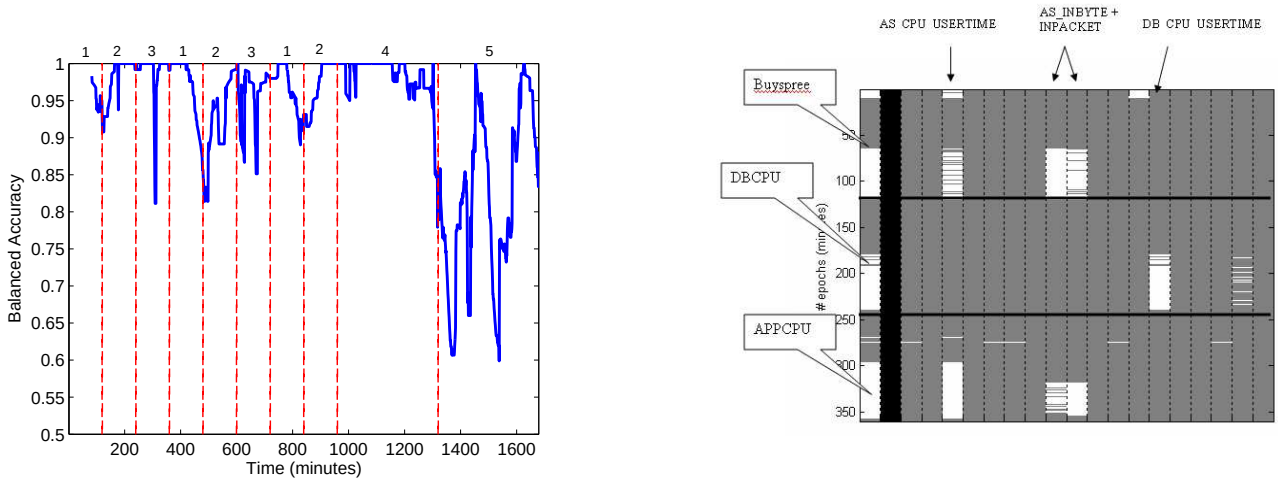
Table 2: Summary of performance results. First three rows show results for ensemble of models with different size training window (80, 120, 240 samples).

of the ensemble of models in those cases where the single model had false alarms, we see that three metrics that were not chosen in the single model contribute to the lower false alarm rate of the ensemble: number of incoming packets and bytes to the application server and the number of requests to the system. Additionally, the first two also appear in the model specific to the BUYSPREE workload. Thus, the ensemble of models is capable of capturing more of the important patterns, both of violations and non-violations by focusing on different metrics for different workloads.

Adaptation to different workloads To show how the ensemble of models adapt to changing workloads, we store the accuracy of the ensemble of models as the ensemble is trained. The changes in the ensemble's accuracy as a function of the number of samples is shown in Figure 3. There are initially no models in the ensemble until enough samples of violations are observed. The ensemble's accuracy remains high until new workloads elicit behaviors different from those already seen, but adaptation is quick once enough samples of the new workload have been seen. An interesting situation occurs during adaptation for the last workload condition (BURST, marked as 5 in the figure). We see that the accuracy decreases significantly when this workload condition first appears, but improves after about 100 samples. It then decreases again, illustrating the complexity of this workload and the need for multiple models to capture its behavior, and finally recovers as more samples appear. It is worth noting that the there is a tradeoff between adaptation speed and robustness of the models, e.g., with small training window, adaptation would be fast, however, the models might not be robust to overfitting and will not generalize to new data.

5.2 Metric attribution

Figure 3 illustrates the metric attribution for part of the test set for one of the ensemble of models we learned. The image illustrates which metrics were implicated with an SLO violation during the different workload conditions. For instances of violations in the BUYSPREE workload, network inbyte and inpacket are implicated with most of the instances of violation. In addition, due to the increased workload, the App server CPU (usertime) is sometimes also implicated with the SLO violations, as the increased workload causes increased CPU load. For the DBCPU workload, we see that the DB CPU is implicated for all instances of SLO violation, while for the APPCPU workload,



(a) (b)

Figure 3: (a) Balanced accuracy of ensemble of models during training. Vertical dashed lines show boundaries between workload changes. Numbers above figure enumerate which of the five types of workload corresponds to each period (1=DBCPU, 2=APPCPU, 3=BUYSPREE, 4=RAMP, 5=BURST). (b)Metric attribution image for the ensemble of models. First column is the actual SLO state (white indicates violation), other columns are the state of each metric chosen in the ensemble (white indicates the violation can be attributed to this metric). Y-axis is epochs (minutes), dark horizontal lines show boundaries between changes in workloads.

all the App server CPU is implicated with all SLO violations. Some SLO violation instances in the APPCPU workload also implicate the network inbyte and inpacket as well, but, the image does not show that in contrast to the BUYSPREE workload, the two metrics are implicated because of *decreased* network activity, as opposed to increased activity in the BUYSPREE workload. With the knowledge of the app server CPU being high, and the network activity being low, a diagnosis pointing to an external contention to the CPU resource can be easily deduced.

It is worth noting that the image illustrates that for different workloads (hence different performance problem causes) there appears to be different "signatures" of metric attribution — cataloging such signatures could potentially serve as a basis for fast diagnosis based on information retrieval principals.

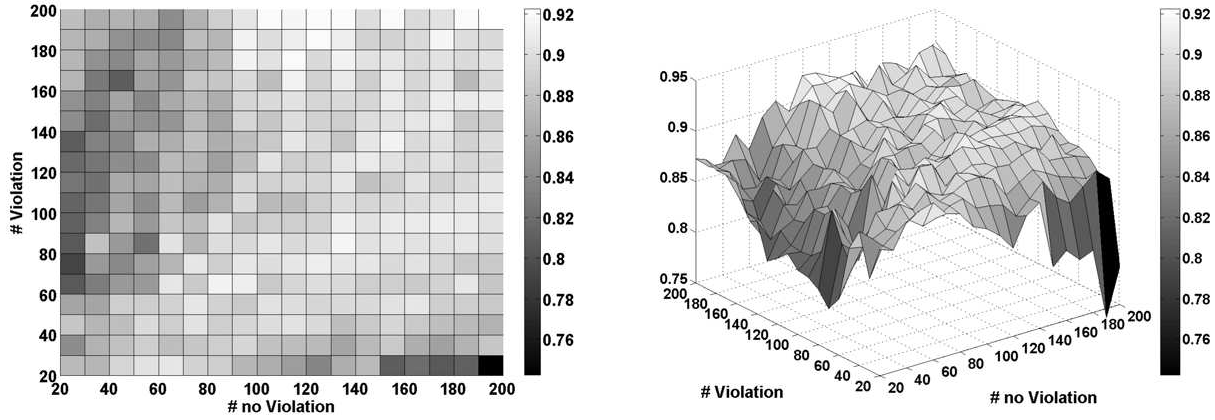
5.3 Learning surfaces: determining the appropriate sample size

To test how much data is needed to learn accurate models we take the common approach of testing it empirically. Typically, in most machine learning research, the size of the training set is incrementally increased and accuracy is measured on a fixed test set. The result of such experiments are learning curves, which typically show what is the minimum training data needed to achieve models that perform close to the maximum accuracy possible. In the learning curve approach, the ratio between violation and non-violation samples is kept fixed as the number of training samples is increased. Additionally, the test set is usually chosen such that the ratio between the two classes is equivalent to the training data. These two choices can lead to an optimistic estimate of the number of samples

needed in training, because real applications (including ours) often exhibit a mismatch between training and testing distribution and because it is difficult to keep the ratio between the classes fixed.

To obtain a more complete view of the amount of data needed, we follow the work of Forman and Cohen [15], and vary the number of violations and non-violations independently of each other, testing on a fixed test set that has a fixed ratio between violations and no violations. The result of this testing procedure is a *learning surface* which shows the performance for any ratio of violations to non-violations. Figure 4 shows the learning surface for the RAMP workload we described in the previous section. Each point in the learning surface is the average of five different trials. What we see visually from the learning surface (both from the top view and side view) is that after about 80 samples of each class, we reach a plateau in the surface, indicating that increasing the number of samples does not necessarily provide much higher accuracies. The surface also shows us that small numbers of violations paired with high numbers of nonviolations results in poor performance, even though the total number of samples is high; e.g., accuracy with 200 nonviolations and 20 violations is only 75%, in contrast to other combinations on the surface with the same total number of samples (220) but significantly higher accuracy.

Table 3 summarizes the minimum number of samples needed from each class to achieve 95% of the maximum accuracy. We see that the simpler APPCPU and DBCPU workloads require fewer samples of each class to reach this accuracy threshold compared to the more complex RAMP or BURST workloads.



(a) Top view (b) Side View

Figure 4: Learning surface for RAMP experiment showing balanced accuracy. The color map on each figure shows the mapping between color and accuracy. Each quad in the surface is the balanced accuracy of the right bottom left corner of the quad.

5.4 Summary of Results

The key observations of our results may be summarized as follows:

1. Even when multiple workload-specific models are trained on separate parts of the workload, the ensemble

Workload	# vio	# no vio	max BA(%)
RAMP	90	80	92.35
BURST	180	60	81.85
BUYSPREE	40	40	95.74
APPCPU	20	30	97.64
DBCPU	20	20	93.90

Table 3: Minimum sample sizes needed to achieve accuracy that is at least 95% of the maximum accuracy achieved for each workload condition.

method results in higher accuracy because the ability to learn and manage multiple models results in better characterization of these complex workloads.

2. The ensemble method also gives better metric attribution than workload-specific models, and allows us to observe that different workloads seem to be characterized by metric-attribution "signatures". Future work may include exploiting such distinct signatures for enhanced diagnosis.

3. The ensemble provides rapid adaptation to changing workloads, but adaptation speed is sensitive not only to the number of samples seen but also the ratio between the two classes of samples (violation and nonviolation).

4. The observed overlap of the metrics chosen by various models in the ensemble suggests that there is some redundancy among the models in the ensemble. One area of future work will be to investigate and remove this redundancy.

5. Choosing a "winner take all" selection function rather than one that combines models in the ensemble gives excellent results; future work may include investigating more complicated selection functions or further refinements of metric attribution, such as ranking the metrics based on their actual log-likelihood ratio difference values.

6. At no point in our process is explicit domain-specific knowledge required, though such knowledge could be used, e.g., to guide the selection of which metrics to capture for model induction. Future work may investigate which aspects of our approach could be enhanced by domain-specific knowledge.

6 Related Work

Automatic monitoring and dynamic adaptation of computer systems is a topic of considerable current interest, as evidenced by the recent spate of work in so-called "autonomic computing". Recent approaches to automatic resource allocation use feedforward control [16], feedback control [17], and an open-market auction of cluster resources [18]. Our goal is to understand the connection between high-level actionable properties such as SLO's, which drive these

adaptation efforts, and the low-level properties that correspond to allocatable resources, decreasing the granularity of resource allocation decisions. Other complementary approaches pursuing similar goals include WebMon [19] and Magpie [20]. Aguilera et al. [21] provide an excellent review of these and related research efforts, and propose algorithms to infer causal paths of messages related to individual high-level requests or transactions to help diagnose performance problems.

Pinpoint [6] uses anomaly-detection techniques to infer failures that might otherwise escape detection in componentized applications. The authors of that work observed that some identical components have distinct behavioral modes that depend on which machine in a cluster they are running on. They addressed this by explicitly adding human-level information to the model induction process, but approaches such as ours could be applied for automatic detection of such conditions and disqualification of models that try to capture a "nonexistent average" of multiple modes. Similarly, Mesnier et al. [22] find that decision trees can accurately predict properties of files (e.g., access patterns) based on creation-time attributes, but that models from one production environment may not be well-suited to other environments, inviting exploration of an adaptive approach.

Whereas "classic" approaches to performance modeling and analysis [23] rely on *a priori* knowledge from human experts, our techniques induce performance models automatically from passive measurements alone. Other approaches perform diagnosis using expert systems based on low-level metrics [24]; such approaches are prone to both false negatives (low-level indicators do not trigger any rule even when high-level behavior is unacceptable) and false negatives (low-level indicators trigger rules when there is no problem with high-level behavior). In contrast, our correlation-based approach uses observable high-level behavior as the "ground truth", and our models are induced and updated automatically, allowing for rapid adaptation to a changing environment.

7 Conclusion

We routinely build and deploy systems whose behavior we understand only at a very coarse grain. Although this observation motivates autonomic computing, progress requires that we be able to characterize actionable system behaviors in terms of lower-level system properties. Furthermore, previous efforts showed that no single low-level metric is a good predictor of high-level behaviors. Our contribution has been to extend our successful prior work on using TAN (Bayesian network) models comprising low-level measurements both for prediction of compliance with service-level objectives and for metric attribution (understanding which low-level properties are correlated with SLO compliance or noncompliance). Specifically, by managing an ensemble of such models rather than a single model, we achieve rapid adaptation of the model to changing workloads, infrastructure changes, and external disturbances. The result is the ability to continuously characterize a key high-level behavior of our system (SLO compliance) in

terms of multiple low-level properties; this gives the administrator, whether human or machine, a narrow focus in considering repair or recovery actions. Using a real system under realistic workloads, we showed that collecting instrumentation, inducing models, and maintaining the ensemble of models is inexpensive enough to do in (soft) real time. Beyond the specific application of TAN models in our own work, we believe our approach to managing ensembles of models will prove more generally useful to the rapidly-growing community of researchers applying machine learning techniques to issues of system dependability.

8 Acknowledgments

We thank Terence Kelly for his help with generating the workloads, and Rob Powers and Peter Bodick for useful comments on the paper.

References

[1] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. Chase, "Correlating instrumentation data to system states: A building block for automated diagnosis and control," in *6th Symposium on Operating Systems Design and Implementation (OSDI'04)*, Dec. 2004.

[2] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[3] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine Learning*, vol. 29, pp. 131–163, 1997.

[4] G. Brier, "Verification of forecasts expressed in terms of probability," *Monthly weather review*, vol. 78, no. 1, pp. 1–3, 1950.

[5] I. Cohen and M. Goldszmidt, "Properties and benefits of calibrated classifiers," in *8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pp. 125–136, Sept. 2004.

[6] E. Kıcıman and A. Fox, "Detecting application-level failures in component-based internet services." Submitted for publication, September 2004.

[7] R. Duda, P. Hart, and D. Stork, *Pattern Classification*. New York: John Wiley and Sons, 2001.

[8] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[9] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1137–1145, 1995.

[10] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[11] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, “Ensemble selection from libraries of models,” in *International conference on Machine learning ICML*, 2004.

[12] P. Domingos, “Bayesian averaging of classifiers and the overfitting problem,” in *International Conference on Machine Learning ICML*, pp. 223–230, 2000.

[13] A. Messinger, “Personal comm..” BEA Systems, 2004.

[14] S. Duvur, “Personal comm..” Sun Microsystems, 2004.

[15] G. Forman and I. Cohen, “Learning from little: Comparison of classifiers given little training,” in *8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pp. 161–172, Sept. 2004.

[16] E. Lassettre, D. Coleman, Y. Diao, S. Froelich, J. Hellerstein, L. Hsiung, T. Mummert, M. Raghavachari, G. Parker, L. Russell, M. Surendra, V. Tseng, N. Wadia, and P. Ye, “Dynamic Surge Protection: An Approach to Handling Unexpected Workload Surges with Resource Actions that have Lead Times,” in *Proc. of 1st Workshop on Algorithms and Architectures for Self-Managing Systems*, (San Diego, CA), June 2003.

[17] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus, “Using control theory to achieve service level objectives in performance management,” *Real Time Systems Journal*, vol. 23, no. 1–2, 2002.

[18] K. Coleman, J. Norris, A. Fox, and G. Candea, “OnCall: Defeating spikes with a free-market server cluster,” in *Proc. International Conference on Autonomic Computing*, (New York, NY), May 2004.

[19] P. K. Garg, M. Hao, C. Santos, H.-K. Tang, and A. Zhang, “Web transaction analysis and optimization,” Tech. Rep. HPL-2002-45, Hewlett-Packard Labs, Mar. 2002.

[20] P. Barham, R. Isaacs, R. Mortier, and D. Narayanan, “Magpie: real-time modelling and performance-aware systems,” in *Proc. 9th Workshop on Hot Topics in Operating Systems*, (Lihue, Hawaii), June 2003.

[21] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, “Performance debugging for distributed systems of black boxes,” in *Proc. 19th ACM Symposium on Operating Systems Principles*, (Bolton Landing, NY), 2003.

[22] M. Mesnier, E. Thereska, G. R. Ganger, D. Ellard, and M. Seltzer, “File classification in self-* storage systems,” in *Proceedings of the First International Conference on Autonomic Computing (ICAC-04)*, May 2004.

[23] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. New York, NY: Wiley-Interscience, 1991.

[24] A. Cockcroft and R. Pettit, *Sun Performance and Tuning*. Prentice Hall, 1998.